MATLAB: Mathematical Uses In Linear Algebra and Differential Equations

Written By Abi Chiaokhiao

Files: W3_StartFile.m

Disclaimer: Some knowledge of linear algebra terms is assumed. Many examples taken from mathworks.com

Table of Contents

Common Math Functions

Function/Keyword	**Desc**	**Function/Keyword**	**Desc**
`exp(#)`	Exponential, e^#	`sqrt(#)`	Square root of #
`log(#)`	Natural log of #	`imag(array)`	Imaginary parts of `array` returned
`log10(#)`	Log base 10 of #	`abs(# or array)`	Absolute value returned of given data (1 # or an `array`)
`nthroot(#,n)`	The `n`th root of #		

Polynomials

Creating and Evaluating at a Point

Initiate polynomial with a vector of coefficients, then call `polyval()`.

Example A vector to represent $p(x)=4x^5-3x^2+2x+33$.

```
p = [4 0 0 -3 2 33];
polyval(p,2)
```

Output

```
ans = 153
```

Integrating and Differentiating

To get a derivative `polyder()` is called and a new vector of coefficients is returned.

Example

```
p = [1 0 -2 -5];
q = polyder(p)
```

Output

```
q = 3 0 -2
```

For the derivative of the product of 2 polynomials, put them both in the arguments.

```
polyder(p1,p2)
```

For the derivative of quotient (polynomial a)/(polynomial b), set the function to have 2 outputs.

```
[q,d] = polyder(a,b)
```

Where `q` would be the coefficient vector of the numerator, and `d` would be the coefficient vector of the denominator.

To integrate the same process is done with `polyint()`.

Polynomials (con.)

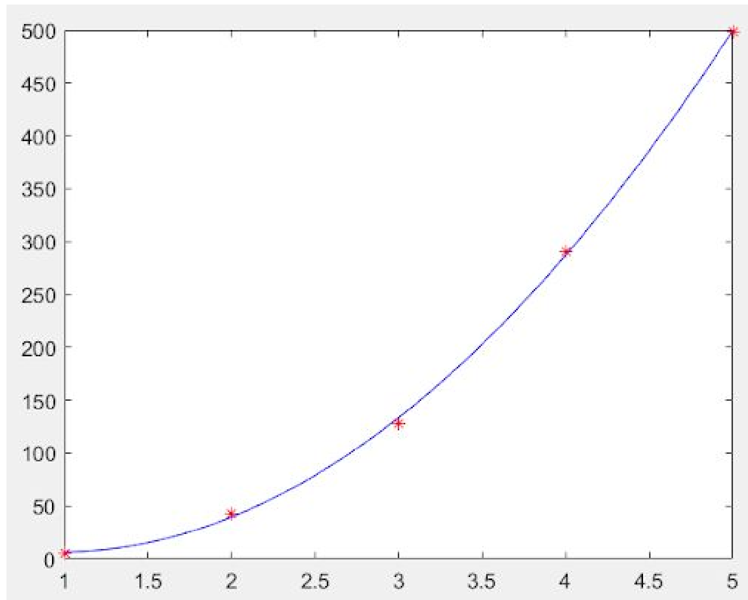
Example

```
p = [4 -3 0 1];
q = polyint(p)
```

```
q = 1×5
    1 -1 0 1 0
```

Fitting a Polynomial to Data

Given x & y data, a polynomial can be estimated with `polyfit()`. After the x & y data are given, the degree desired is the 3rd input.



Example

```
x = [1 2 3 4 5];
y = [5.5 43.1 128 290.7 498.4];
p = polyfit(x,y,3)
x2 = 1:0.1:5;
y2 = polyval(p,x2);
plot(x,y,'r*',
     x2,y2,'b-')
```

Output

```
p = -0.1917   31.5821   -60.3262   35.3400
```

Linear Algebra/Matrices

Basic Matrix functions

Function/Keyword	Desc	Function/Keyword	Desc
`transpose(m)`	Transpose of given matrix (switches columns & rows)	`cross(A,B)`	Takes cross product of matrices `A` and `B`
		`mpower(m,#)`	Takes the matrix `m` and takes it to the power of `#`

Eigenvalues/Eigenvectors

When a matrix is multiplied by a vector, the result can be split into a coefficient multiplied by the original vector. The coefficient is the eigenvalue, and the vector is the eigenvector. To get the same result the matrix can then be replaced by the eigenvalue.

Linear Algebra/Matrices (con.)

Eigenvalues/Eigenvectors (con.)

Example

$$\begin{pmatrix} 1 & 2 & 0 \\ 2 & 1 & 0 \\ 0 & 0 & -3 \end{pmatrix} \times \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 3 \\ 3 \\ 0 \end{bmatrix} = 3 \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}$$

The eigenvalue is **3**

The eigenvector is **[1; 1; 0]**

`eig()`

To get Eigenvalues for a few eigenvectors you do `eig(matrixName)`. By itself a vector of eigenvalues will be given, if assigned to have two outputs P & D, this will assign P to being a matrix where each column is an eigenvector, and D to be a diagonal matrix (use `diag(D)` to get a vector of the diagonal elements) where the diagonal is each corresponding eigenvalue.

Example

```
A = [1 2 3; 4 5 6; 7 8 9];
eig(A) % Output: [16.1168; -1.1168; 0]
[P,D]=eig(A)
% Output: P = [-0.2320 -0.7858 0.4082;
             % -0.5253 -0.0868 -0.8165;
             % -0.8187  0.6123  0.4082]
        % D = [16.1168      0      0;
             %    0    -1.1168    0;
             %    0         0      0]
```

Solving Systems of Equations (*Decomposition*)

There's many times where you can are solving for many variables but it would be difficult to solve for each of them individually. By turning these systems into matrices, the variables can be solved for quickly. Solving these systems is *Decomposition*.

Technique 1: Using Reduced Row Echelon Form (RREF) Matrices ~ Very long, the manual way

Example (long)

```
B = [2 3 -4 1 2;3 -1 -1 2 4;1 -7 5 -1 6]
% Row Echelon Form (REF)
B(1,:)=(1/2)*B(1,:);
B(2,:)=B(2,:)-3*B(1,:);B(3,:)=B(3,:)-B(1,:);
B(2,:)=(-2/11)*B(2,:);
B(3,:)=B(3,:)+(17/2)*B(2,:);
B(3,:)=(-11/8)*B(3,:);
```

Output

```
B =  1 3/2    -2      1/2      1
     0  1   -10/11  -1/11   -2/11
     0  0      1      25/8   -19/4
```

```
% RREF
B(1,:)= B(1,:) -   (3/2)   *  B(2,:);
B(2,:)= B(2,:) + (10/11)  *  B(3,:);
B(1,:)= B(1,:) +   (7/11)  *  B(3,:);
```

Output

```
B =  1  0  0  21/8  -7/4
     0  1  0  11/4  -9/2
     0  0  1  25/8  -19/4
```

```
% Final answer in column vector form
% Let x4 = t ⇒
[x1;x2;x3;x4]
= t*[21/8;11/4;25/8;1]
+ [-7/4;-9/2;-19/4;0]
```

Using `rref()`

```
B = [1 7 9 11; 13 1 4 2; 4 -3 15 2];
rref(B)
```

Output (4th column = values of each unknown variable)

```
B =  1  0  0  -39/886
     0  1  0  467/422
     0  0  1  181/494
```

Solving Systems of Equations (con.)

Technique 2 (all the same using different commands)**: No Simplifying of A{x} = {b}**
Where A & b correspond to a matrix and vector in the system [A]{x} = {b}

Example

```
B = [1 0 3 12; 3 2 0 32;0 -3 2 2];
rrefSolve = rref(B)
A = B(:,1:3)
b = B(:,end)
linSolve = linsolve(A,b)
invSolve = inv(A)*b
othSolve = A\b
```

Output

```
A =    1   0   3
       3   2   0
       0  -3   2
b =   12  32   2
rrefSolve = linSolve = invSolve = othSolve =
1 0 0   252/23
0 1 0   -10/23
0 0 1     8/23
```

Note: some matrix systems will give slightly different answers using `rref()`, most likely due to being larger or more complicated.

There's also lu which I didn't cover, and other methods are here. More on linear algebra that I didn't include in this tutorial is here.

Ordinary Differential Eqns (ODEs)

A differential is like a derivative of a function, but added in changes with respect to the independent variable. So if f(x) is the original function, the differential dy = f'(x)dx, dx being a change in x. Two types are *stiff* and *nonstiff*, the former meaning it can only be solved with a small step size.

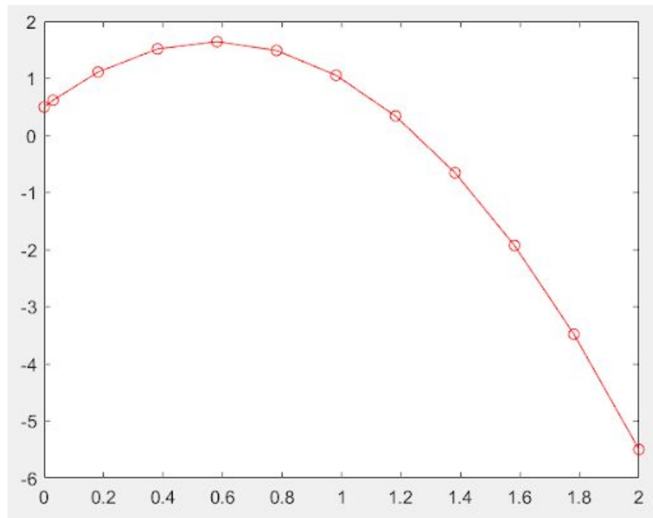
Stiff - `ode23s()`

To solve a stiff ODE, give `ode23s()` the *function*, the *range of the independent variable*, and the *initial condition*.

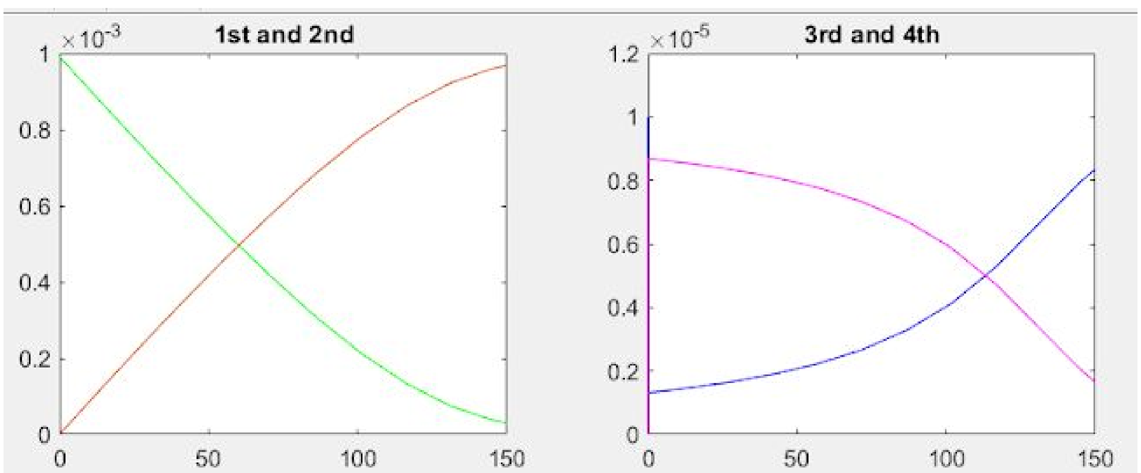
Example

```
tspan = [0 2];
y0 = 0.5;
[t,y] = ode23s(@(t,y) -7*t+4, tspan, y0);
plot(t,y,'o-r')
```

Note: the `@` denotes a function, the independent variables are denoted by `(t,y)`. More info here.



Higher order ODEs (using `ode23s()`, in W3_BonusFile.m)



Ordinary Differential Eqns (con. 1)

Higher order ODEs (con.)

Example

```
k = [1e9 1.5e5 1];
% An extra vector which are each a variable used in
% the function later
c0 = [1e-5 1e-3 0 0];
% Corresponding to multiple initial conditions
tspan = [0 150];
[t, c] = ode23s(@odeSolve, tspan, c0,[], k);
subplot(1,2,1)
plot(t,c(:,2),'-g',t,c(:,3))
title('1st and 2nd');
subplot(1,2,2)
plot(t,c(:,1),'-b',t,c(:,4),'-m')
title('3rd and 4th');
function dydt = odeSolve(t,c,k)
     % r1, r2, r3 are variables
     % calculated using constants
     % from the initial condition
     % and the other given constants in k
     r1 = k(1)*c(1)*c(2);
     r2 = k(2)*c(4);
     r3 = k(3)*c(4);
     % CS, CE, CES, CP are the equations
     % dependent on r1, r2, r3
     CS = -r1+r2;
     CE = -r1+r2+r3;
     CES = r1-r2-r3;
     CP = r3;
     dydt = [CE; CS; CP; CES];
     % Denotes each new point calculated into a matrix
end
```

There's also `ode23t()` which uses the trapezoidal method.

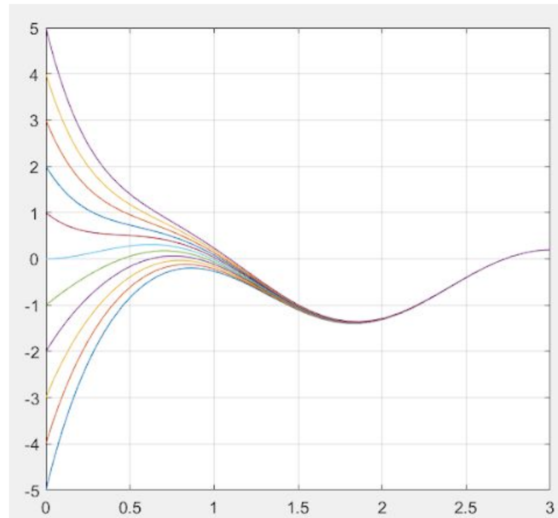
Ordinary Differential Eqns (con. 2)

Nonstiff - `ode45()`

In theory, `ode23s()` and `ode23t()` could most likely solve a nonstiff as they are less particular, but in case a nonstiff function is specifically wanted, `ode45()` can be used.

Example

```
dy = @(t,y) -3*y + 5*cos(2t).*sin(t);
y0 = -5:5;
tspan = [0 3];
[t,y] = ode45(dy,tspan,y0);
plot(t,y)
grid on
```



Run in matlab, change y0 to show effects

Note: I have the function defined at the top so it doesn't need to be in the function call.

More on `ode45()` here. If `ode45()` doesn't work, another nonstiff option is `ode23()`.

Functions (pg 1/2)

Input	Output	Description	Section
`polyval(p,#)`	Vector	Evaluates a polynomial coefficient vector `p` at `#`	Polynomials
`polyder(a,b)`		Returns derivative (same form as inputs) of polynomials (if 1 polynomial coefficient vector just that, if 2, the derivative of those polynomials' product)	
`polyint(p)`		Returns integral of polynomial coefficient vector `p`	
`polyfit(x,y,n)`		Fits to `n`th degree a polynomial from the x & y data	
`transpose(m)`	Matrix	Transpose of matrix `m`	Linear Algebra/Matrices
`mpower(m,#)`		Takes `m` and takes `#` the matrix to that power	
`cross(A,B)`		Takes the cross product `AxB`	
`eig(m)`	Vector or 2 matrices (if set to to output)	A vector of eigenvalues for a few eigenvectors of matrix `m` if only one output. If assigned to have two outputs the first output will be a matrix where each column is an eigenvector, and the second a diagonal matrix where the diagonal is each corresponding eigenvalue.	
`diag(m)`	Vector	The diagonal of a matrix `m` is returned	
`rref(m)`	Matrix	The reduced row echelon form of that matrix to aid in solving for variables.	
`linsolve(A,b)`	Vector	A vector where each element represents a variable from the columns of matrix `A`	

Functions (pg 2/2)

Input	Output	Description	Section
`inv(A)`	Matrix	The inverse of a matrix `A` is taken	Linear Algebra/Matrices
`ode23s(func, tspan, y0)`	Vectors	For stiff (needing small step) ODEs, `func` is the function being solved, `tspan` is the span of the independent variable, and `y0` is the initial condition(s). Returns vectors corresponding to points that can be plotted.	Ordinary Differential Eqns (ODEs)
`ode45(func, tspan, y0)`		For nonstiff ODEs, `func` is the function being solved, `tspan` is the span of the independent variable, and `y0` is the initial condition(s). Returns vectors corresponding to points that can be plotted.	